carnd_t2_p4_pid_controller

Carnd - term 2 - project 4 - pid controller

Overview

The goal of this project is to implement an PID Controller to maneuver the vehicle around the track in the simulator. The simulator will provide cross track error and velocity at each time step for calculating the necessary steering angel. It requires tuning the controller parameters: tau_p, tau_i, and tau_d, so that the vehicle will drive safely as well as staying on track.

Project Repository

All resource are located in Udacity's project repository CarND-PID-Control-Project (https://github.com/udacity/CarND-PID-Control-Project)

Project Submission

All modified code including results are committed to my personal github page carnd_t2_p4_pid_controller (https://github.com/chriskcheung/carnd_t2_p4_pid_controller)

Key Files

main.cpp

establishes communication between simulator and PF model using uWebSocketIO, and reads in data during set time interval and send sensor measurements of cross track error, cte, and velocity of vehicle to PID::Init(), PID::UpdateError(), and PID::TotalError() in PID.cpp for processing

PID.cpp

contains 3 main functions: PID::Init(), PID::UpdateError(), PID::TotalError().

PID::Init() simply initializes all errors parameters: p_error, i_error, and d_error, to zeros and assigns tau parameters: Kp, Ki, and Kd, to manually tuning inputs from running pid program.

PID::UpdateError() defines each error using the provided cte and previous cte:

```
d_error = cte - p_error;
p_error = cte;
i_error += cte;
```

PID::TotalError() calculates the total error using the following equation:

```
return Kp*p_error + Ki*i_error + Kd*d_error;
```

Implementation Challenge

This project is very striaght forward with provided cte and the algorithm needed to calculate the steering angle. In here, I will focus on the tuning strategy discussion instead.

As the lecture taught about using Twiddle to determine the best tau parameters for the controller. I tried to implement that as well, but my implementation seemed to be off and was not able to converge to the proper tau values. Manual tuning seemed to be faster than figuring out the bug in my twiddle implementation. At the end, I manually tuned the controller instead.

I started with tuning tau_p, or Kp, to allow the vehicle to gradually converge to the center of the road. I started with Kp=-0.5 which oscillating too fast. As it dropped from -0.5, the oscillation slowed down. I stopped at kp=-0.05 as the oscillation was slow enough but able to allow the vehicle to enter the first turn.

The next was tuning tau_d, or Kd, to counter the overshoot caused by oscillation. The larger the Ki value, the sharper the counter turn it was. I started from -5.0 and worked my way down to -0.5 and observed that the counter action became smoother at the lower value.

The last parameter tau_i, or Ki, is for compensating the systematic bias of the vehicle such as mechanical alignment bias to a certain degree. It was not obvious that this simulator had bias, but it is always better to assume it has and tune the Ki parameter as part of consideration. The Ki value seemed to be very sensitive as its error was sum over time. I started with a very small number at -0.001, and worked its way up.

After a lot of trial and error, the best results was provided with Kp=-0.1, Ki=-0.005, Kd=-0.9.

Result

The vehicle was able to stay on track for 4 laps. Here are some of the snapshot when vehicles at different turns.

In [ ]: